I C A S E R E P O R T

EMPIRICAL COMPARISON OF PARTITIONED AND

NON-PARTITIONED BUFFER MANAGEMENT

IN VIRTUAL MEMORY SYSTEMS

Richard S. Brice

Stephen W. Sherman

Report No. 76-9

March 25, 1976

(NASA-CR-185728) EMPIRICAL COMPARISON OF PARTITITIONED AND NON-PARTITITIONED BUFFER MANAGEMENT IN VIRTUAL MEMORY SYSTEMS (ICASE) 16 p

N89-71337

Unclas

00/60 0224347

INSTITUTE FOR COMPUTER APPLICATIONS

IN SCIENCE AND ENGINEERING

Operated by the

UNIVERSITIES SPACE RESEARCH ASSOCIATION

at

NASA'S LANGLEY RESEARCH CENTER

Hampton, Virginia

EMPIRICAL COMPARISON OF PARTITIONED AND NON-PARTITIONED BUFFER MANAGEMENT IN VIRTUAL MEMORY SYSTEMS

Richard S. Brice

Department of Civil, Mechanical and Environmental Engineering
George Washington University

Stephen W. Sherman*

Institute for Computer Applications in Science and Engineering

A B S T R A C T

Buffer pools are created and managed in data base systems in order to reduce the total amount of accesses to secondary memory. In systems using virtual memory the virtual buffer pools can increase secondary memory accesses by increasing paging in the system. In this paper we compare the performance of systems where the virtual buffer is allocated fixed amounts of real storage and partitioned from the program to the performance of systems where the virtual buffer and program compete for real memory. Our analysis utilizes empirical data gathered in a multifactor experiment. The factors we consider are memory size, virtual buffer size, virtual page replacement algorithm, buffer management algorithm and size of real memory allocated to the virtual buffer.

* On leave from the University of Houston

This paper was prepared as a result of work performed under NASA Contract No. NAS1-14101 while the second author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665. The work of the first author was supported by NASA Grant NGR-09-010-078

1. INTRODUCTION

Computer programs that require substantial amounts of I/O often use part of primary memory as a buffer for data from secondary memory. If the overhead to manage the data in primary memory is negligible and the buffer consumes previously unused primary memory, then the use of buffers improves performance due to the faster access to primary than to secondary memory. The use of a buffer in a virtual memory system may cause a decrease in performance due to competition for primary memory between the program and the buffer. Performance can also be degraded by double paging. The dynamics of double paging was characterized by Goldberg and Hassinger [1] as the running of a paged operating system under a paged virtual machine monitor. In this paper, double paging refers to the management of buffer storage under the control of a paged virtual memory system.

We define virtual buffers as the I/O buffers in a program running on a virtual memory system. In two sets of experiments [2,3] we varied a set of control factors to study their effect on the performance of a system using virtual buffers. The control factors common to both experiments were virtual buffer manager, virtual buffer size, primary memory size and paging replacement algorithm. The first set of experiments [2] used a global page table for paging and allowed real pages to migrate between program and buffer. The second set of experiments [3] partitioned memory between the program and buffer isolating the program from the buffer paging and the buffer from the program paging. We refer to the first set of experiments as the non-partitioned experiments and the second set as the partitioned experiments. Both sets of experiments were conducted in a controlled laboratory environment [4] by running a data base management program on a dedicated system and measuring the performance as we varied the factors. The data base management program executed a predetermined and unvarying script.

In this paper we compare the performance of the partitioned and non-partitioned experiments. We define performance in terms of system I/O (paging in the system plus I/O accesses in the buffer) and quantify the contributions of the components of system I/O to the differences in performance. We show that partitioning the real memory between buffer and program can produce slightly better performance than a non-partitioned system but usually produces performance that is noticeably worse. The components that cause most of the differences in performance between the partitioned and non-partitioned systems are program paging and double paging. The program paging in the non-partitioned system is significantly less than the program paging in the partitioned system when the virtual buffer size is large. There is significantly less double paging in the non-partitioned system than in the partitioned system when the virtual buffer size is small.

2. ENVIRONMENT

The partitioned and non-partitioned experiments were conducted on a PRIME 300 minicomputer. The PRIME 300 has a 16-bit word size and supports up to 256K words of real memory (1K=1024 words). Our system has 64K words. The PRIME has virtual memory hardware which supports up to 512 pages of 512 words each. The PRIME peripherals of interest in these experiments consist of two moving head disks each having a capacity of 3 million words.

We have instrumented the PRIME's operating system with a software probe. The probe is locked in memory and cannot interact with the paging process. The probe records events that cause a significant change in the system such as the occurrence of a page fault. The application program we used in our experiments is a prototype data base management (DBM) system. The DBM system is run on a dedicated machine with the software probe collecting significant events on tape for later analysis. The DBM system executes the same script of data base requests in each of the experiments. The script completely traverses the data base and causes reading, insertion and deletion of data as it is executed.

The DBM system organizes data in a tree structured format. Requests are made in a series of primitive functions that perform elementary operations on the data base. The data, names and pointers in the data base are encoded into 40-word segments. One physical disk record contains eleven data base segments. The data base used in the experiments contains 45 records. We allocate each disk record to a separate page in the virtual buffer to avoid physical page boundary overlaps. The DBM virtual buffer size is chosen by the user when the DBM system is initiated. The size of the virtual buffer can range from 1 to 64 pages in 1 page increments.

3. PREVIOUS STUDIES

Tuel [5,6] originally studied the paging and I/O performance of an IBM data base system (IMS Version 2.4 system running on VS/2 Release 1.6). Tuel postulated a theoretical model of the buffer I/O (paging in the buffer + I/O accesses). His model and empirical results were reasonably close when only the buffer paging was considered. The early version of IMS used in the study forced the virtual buffer to be searched at every I/O request since a pointer array describing the contents of the buffer was not used. Tuel concluded that buffer I/O increases with increasing virtual buffer size if the buffer is allowed to page. Therefore, the least amount of buffer I/O is achieved by reducing the virtual buffer size to fit the number of pages available in real memory.

Our data base system and later versions of IMS now have a pointer array describing the contents of the virtual buffer. Searching the pointer array causes much less paging activity than the early IMS technique of reading the information from the individual buffers. In a previous paper by Sherman and Brice [2] we assume that searching the pointer array generates no page faults. We develop a very simple theoretical model which predicts total I/O per data base request in the virtual buffer as a function of virtual buffer size in pages, pages of real memory allocated to the virtual buffer and the number of pages in the data base. The model assumes that the random (RAND) page replacement algorithm and RAND buffer manager are used and the data base requests are uniformly distributed.

By use of this model, we show that it is possible to increase the virtual buffer size and reduce the total I/O per data base request in the buffer for various fixed values of real memory allocated to the virtual buffer and the number of pages in the data base.

The factors we considered in the non-partitioned experiments were memory size, virtual buffer size, page replacement algorithm and buffer management algorithm. Our analysis of the empirical results of the non-partitioned experiments led to the following conclusions. The interaction between the paging algorithm and the buffer algorithm did not have a significant effect on the buffer I/O. The RAND buffer manager and the RAND paging algorithm had lower double paging rates than the other managers and the SCH paging algorithm. The cost of system I/O (buffer I/O + paging in the program) can decrease when the virtual buffer size is increased but the amount is very dependent on the amount of real memory available. The performance advantages of virtual buffers can overcome the costs of double paging and the increased program paging resulting from the use of virtual buffers.

The number of factors in our non-partitioned experiments were extended in our partitioned experiments to include five levels for the size of the memory partition allocated to the virtual buffer. Our analysis of the empirical results of the partitioned experiments led to the following conclusions. The size of the buffer partition can cause significant variation in system I/O due primarily to increased program paging. There is a significant double paging effect in the buffer. The partitioned experiments showed that it is possible for the use of a virtual buffer to improve performance in a partitioned system. The chances for improving performance are increased if: (1) The cost of I/O accesses are greater than the cost of paging. (2) The RAND paging algorithm is used in the buffer partition. (3) The SCH paging algorithm is used in the program partition. (4) The RAND buffer manager is chosen. (5) The virtual buffer size is significantly larger than the buffer partition size.

4. EXPERIMENTS

The comparison in this paper is based on those factors which were common to the partitioned and non-partitioned experiments. In both sets of experiments we use the RAND and SCH (second chance [7] also known as the Multics [8] or use bit [9] algorithm) page replacement algorithms. In the non-partitioned experiments the page replacement algorithms do not differentiate between the program and buffer pages. In the partitioned experiments, pages in one partition are not considered by the page replacement algorithm when a fault occurs in the other partition. The changes in the page replacement algorithms to allow partitioning are the only differences in the partitioned and non-partitioned experiments. Both sets of experiments use the FIFO (first in - first out), RAND, SCH, and LRU (least recently used) virtual buffer managers and virtual buffer sizes of 1, 5, 10, 15, and 20 pages. The total amount of real memory is set to 40K, 44K, and 48K words. The buffer partition sizes for the partitioned experiments are 1, 5, 10, 15, and 20 pages.

In the non-partitioned experiments [2] a total of 240 experiments were performed and analyzed. The different experiments were defined by combination of 3 paging algorithms, 4 buffer managers, 4 real memory sizes and 5 virtual buffer sizes. In the partitioned experiments [3] a total of 360 experiments were performed and analyzed. The different experiments were defined by combinations of the 2 paging algorithms, 4 buffer managers, 3 real memory sizes, 5 memory partition sizes and virtual buffer sizes where the virtual buffer was at least as large as the buffer partition size.

5. RESULTS

The complete data for the partitioned and non-partitioned experiments is available in previous papers by Sherman and Brice [2], [3] and is not presented again in this paper. In order to omit congestion we usually present figures that are representative rather than exhaustive in terms of the number of levels of factors available. A complete set of figures for the partitioned experiments can be constructed from the data in Table I, Table II, and Table III in [3]. A complete set of figures for the non-partitioned experiments can be constructed from Table I and the Appendix in [2].

The I/O and paging in the experiments can be organized hierarchically as shown in Figure I. System I/O and I/O access for the partitioned and non-partitioned experiments can be compared directly. The other components in Figure I are influenced by the partition size which is known in each of the partitioned experiments and varies during each of the non-partitioned experiments.

System I/O is shown in Figures II through IV for the non-partitioned experiments and for the partitioned experiments with all the partition sizes chosen for our experiments. The system I/O for the partitioned experiments typically brackets the system I/O for the non-partitioned experiments. The system I/O for the non-partitioned experiments closely approximates the best performance (least system I/O) of the partitioned experiments. Our results are similar to the results by Denning and Spirn [10] which show that the performance of a multiprogramming system with fixed partitions is generally worse than the performance of a system with variable partitions.

<u>System I/O</u>: Components are Program Paging and Buffer I/O
- Program Paging is a function of CV 1, 3, 4 or 1, 4A
- <u>Buffer I/O</u>: Components are I/O Accesses and Buffer Paging
  - I/O Accesses is a function of CV 2, 3, 5
  - <u>Buffer Paging</u>: Components are Double and Reference Paging
    - Double Paging is a function of CV 1 through 5
    - Reference Paging is a function of CV 1 through 5

<u>CONTROL VARIABLES (CV)</u>
1. Paging Algorithm
2. Buffer Manager
3. Virtual Buffer Size
4. Real Memory Size

4A. Partition Size

5. Distribution of Data Base Requests

Fig. I. Hierarchy of I/O and Paging

In our partitioned experiments the same paging algorithm is used in the program partition and the buffer partition. The performance of each partition was analyzed separately. The SCH paging algorithm typically performed significantly better than the RAND paging algorithm in the program partition. The RAND paging algorithm performed only slightly better than the SCH paging algorithm in the buffer partition.

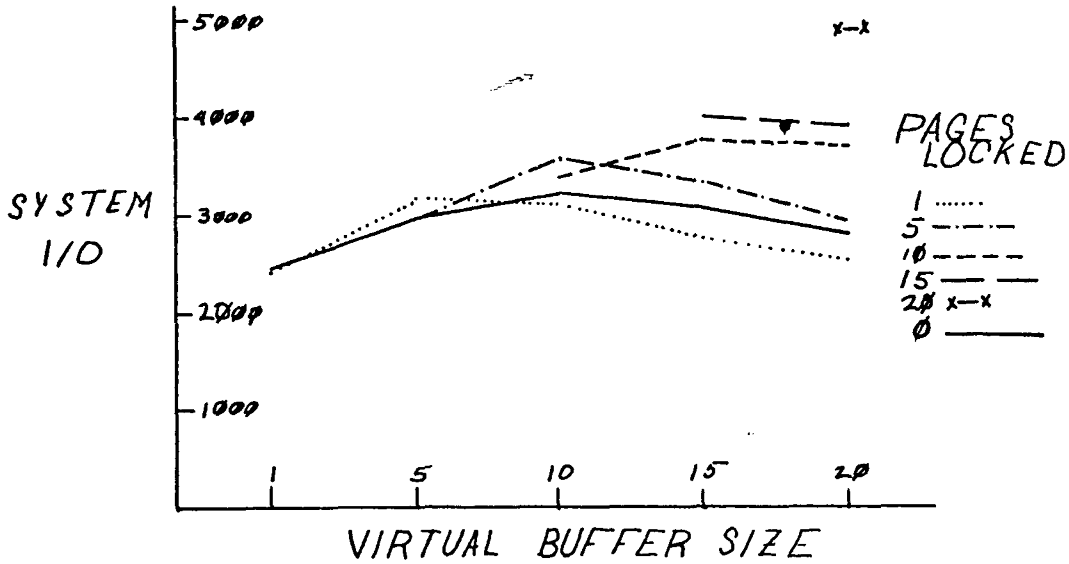

Fig. II. System I/O for partitioned and non-partitioned (0 pages locked) experiments using SCH page replacement algorithm, RAND buffer manager and 40K memory.

An analysis of the page references for the program is shown in Figure V and VI and an analysis of the page references for the data base is shown in Figure VI. Figure V shows the number of page faults that would have occurred in the program partition using the LRU page replacement algorithm for a program partition size of 40 to 95 real pages. The partitioned experiments we conducted had a program partition that ranged from 60 to 95 real pages and the page references analysis for this range is repeated in Figure VI.

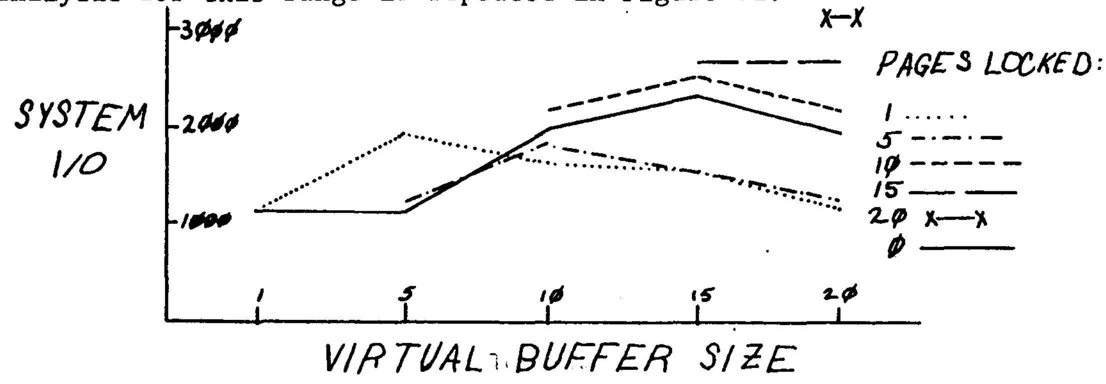

Fig. III. System I/O for partitioned and non-partitioned (0 pages locked) experiments using SCH page replacement algorithm, RAND buffer manager and 44K memory.

Figure VI also shows the number of page faults that would have occurred in the buffer partition using the LRU page replacement algorithm and assuming the complete data base was in the virtual buffer. The data used to construct Figures V and VI was originally gathered to parameterize a set of simulation experiments and the method we used to collect the data is explained in [11]. Figures V and VI illustrate the potential for the program paging component of system I/O to be dominant.

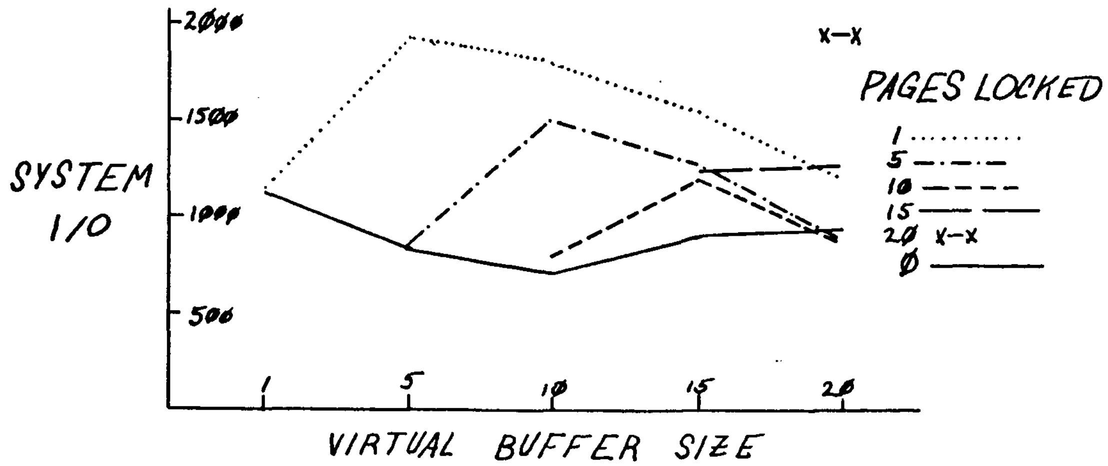

Fig. IV. System I/O for partitioned and non-partitioned (0 pages locked) experiments using SCH page replacement algorithm, RAND buffer manager and 48K memory.

In the non-partitioned experiments system I/O using the SCH paging algorithm was significantly less than system I/O using the RAND paging replacement algorithm. For the partitioned experiments we noted that system I/O would be reduced if the RAND paging algorithm were used in the buffer partition and the SCH paging algorithm used in the program partition. We refer to this system as the optimal partitioned system. The optimal partitioned system offers only a slight improvement in the system I/O from the partitioned system with the SCH algorithm in each partition. A comparison of the optimal partitioned system with the non-partitioned system using the SCH paging algorithm produces figures very similar to Figures II - IV. The performance of the non-partitioned system closely approximates the best performance of even the optimal partitioned system.

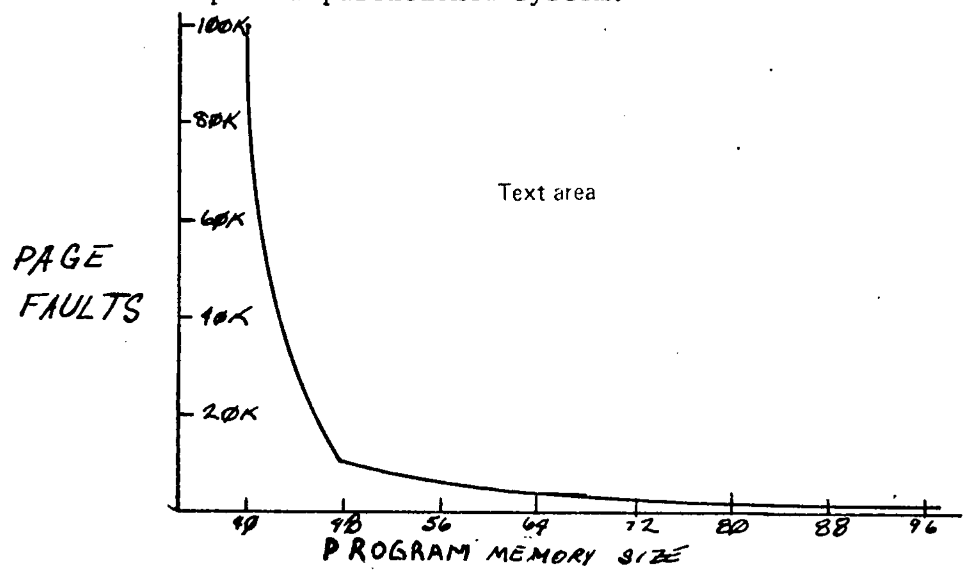

Fig. V PROJECTED PROGRAM PAGING WITH LRU PAGE REPLACEMENT ALGORITHM

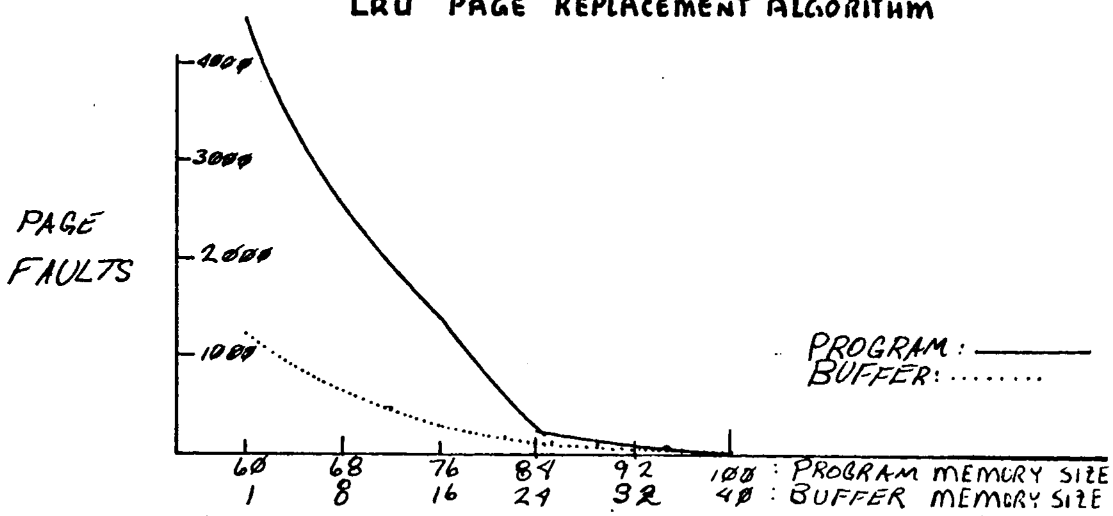

Fig. VI PROJECTED PROGRAM AND BUFFER PAGING WITH LRU PAGE REPLACEMENT ALGORITHM

Buffer I/O, a component of system I/O as shown in Figure I, consists of paging in the buffer and I/O accesses. The I/O accesses do not depend on the partition or the real memory available and are exactly the same for the partitioned and non-partitioned cases. The number of I/O accesses is shown in Figure VII. The RAND buffer manager has significantly fewer I/O accesses at a virtual buffer size of 10 and 15 pages. As was shown in [2], the RAND buffer manager has fewer I/O accesses due to the record reference pattern which contains a few highly referenced records usually separated by strings of references.

Virtual Buffer	Buffer Managers			
Sizes in Records	LRU	SCH	RAND	FIFO
1	1075	1075	1075	1075
5	784	791	774	794
10	684	687	559	688
15	418	422	270	434
20	93	98	154	103

Fig. VII. Number of I/O requests to read records into the virtual buffer

The other component of buffer I/O is buffer paging. Buffer paging consists of double paging and reference paging. Double paging occurs when the buffer manager chooses to replace the information in a page that is not in real memory. Reference paging refers to page faults caused by attempts to use information that is in the virtual buffer but not in real memory. We define the double paging rate to be the number of double page faults divided by the number of I/O accesses. We define the reference paging rate to be the number of reference faults divided by 1075 minus the number of I/O accesses. The maximum number of I/O accesses which can occur in our experiments is 1075.

Figures VIII and IX contain double paging rates for SCH and RAND paging algorithms for the non-partitioned experiments. In the non-partitioned experiments the double paging rates are increasing functions of virtual buffer size and decreasing functions of real memory size. The RAND buffer manager has a noticeably lower double paging rate than the other buffer managers. The RAND page replacement algorithm also has a lower double paging rate than the SCH page replacement algorithm.

Figures X and XI contain the double paging rates for the SCH and RAND paging algorithm for all partition sizes in the partitioned experiments. In the partitioned experiments the double paging rates are also an increasing function of virtual buffer size and a decreasing function of real memory partition size. However, the increase

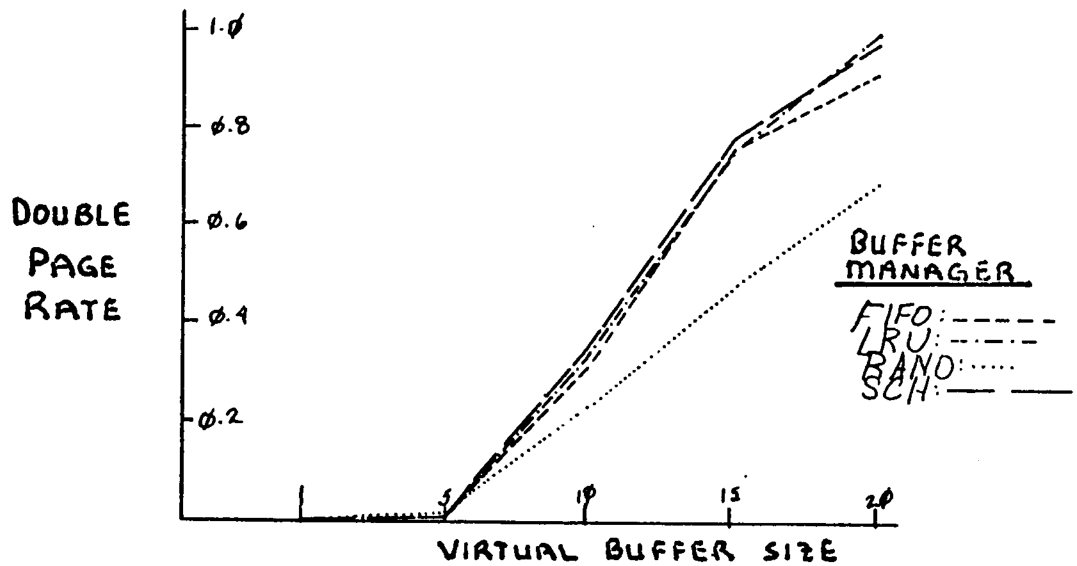

Fig. VIII Double Paging Rate for SCH Paging Algorithm Using 44 K of Real Memory

due to virtual buffer size is much sharper than for the non-partitioned experiments. The RAND buffer manager incurs a significantly lower double paging rate than the other buffer managers in the partitioned experiments. The RAND paging algorithm also incurs a lower double paging rate than the SCH paging algorithm. When the real memory allocated to the buffer is equal to the virtual buffer size the double paging rate is zero.

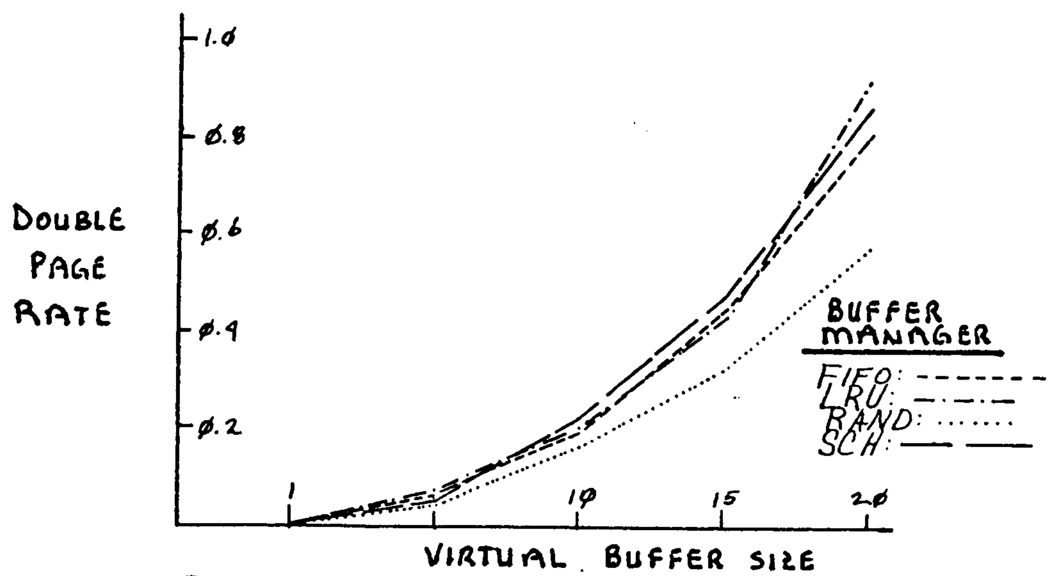

Fig. IX Double Paging Rate for RAND Algorithm Using 44K Real Memory

Our analysis of the non-partitioned experiments produced the average number of real pages allocated to the virtual buffer. Some typical averages for virtual buffer sizes 1, 5, 10, 15, 20 virtual pages are 1, 4.5, 8.4, 10.3, 11.2 real pages. The low double paging rates of the non-partitioned experiments indicate that a considerable portion of the I/O accesses occurred when the virtual buffer had all of its pages in real memory. The partitioned experiments show that the double paging rates of the non-partitioned experiments would have been much higher if the real memory allocated to the virtual buffer was always less than the virtual buffer size.

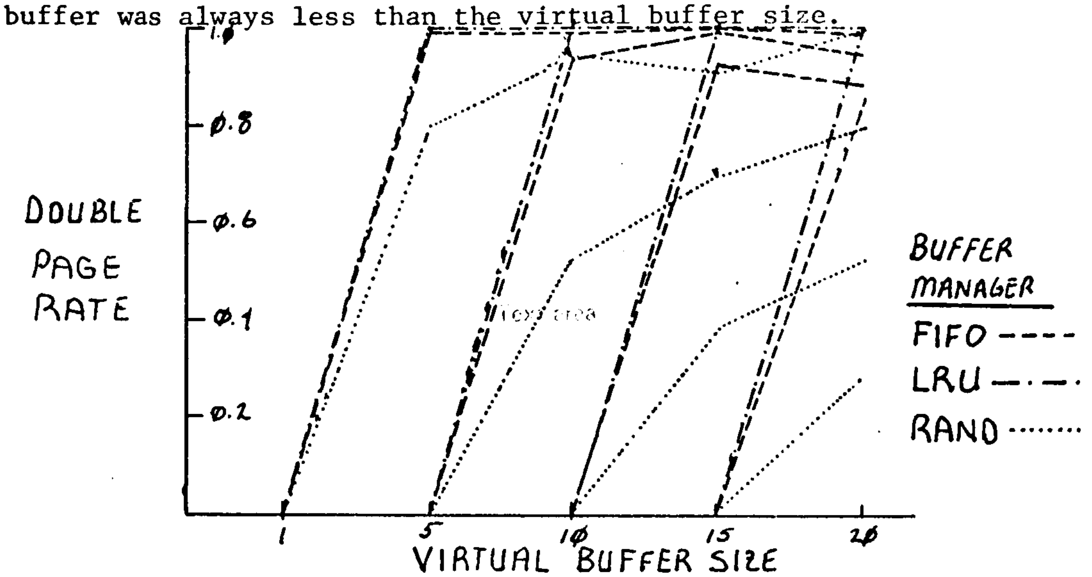

Fig. X. Double paging rate for the SCH paging algorithm in the buffer partition.

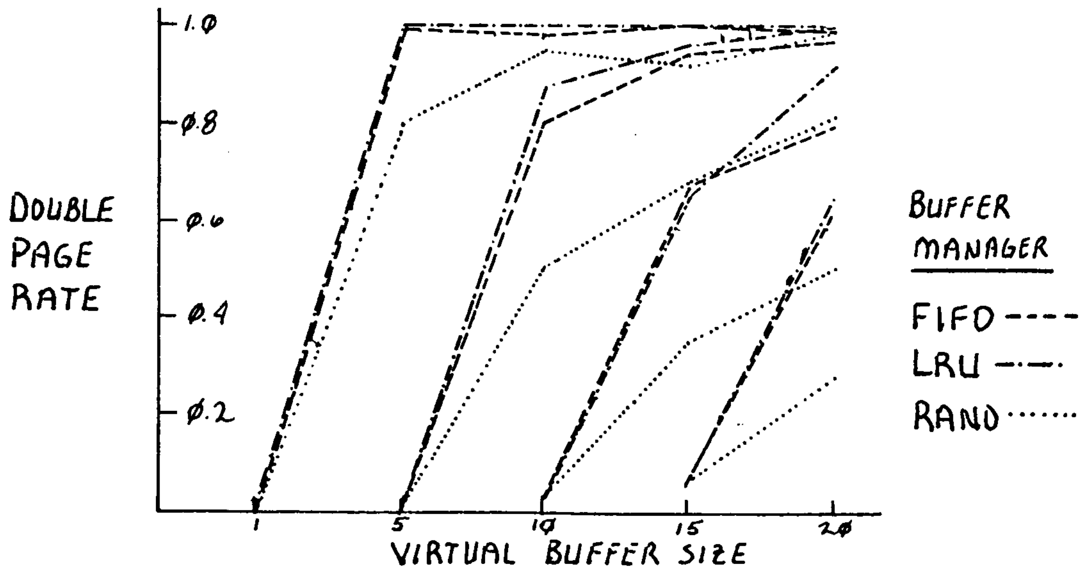

Fig. XI. Double paging rate for the RAND paging algorithm in the buffer partition.

In the non-partitioned experiments we calculated the mean and standard deviation for the number of program faults which occurred between successive buffer faults. The means were a decreasing function of virtual buffer size and the standard deviations were typically larger than the mean. A set of density functions for program faults occurring between successive buffer faults is shown in Figure XII for the five virtual buffer sizes. The small medians of the density functions indicate that the real memory requirements of the virtual buffer occur in bursts. We observed that the data base manager often requests strings of records in order to satisfy some particular data base request. The data is then processed by the program. A string of record requests causes double page faults to occur until the entire virtual buffer is in real memory. Once the virtual buffer is in real memory, any remaining requests in the string do not cause double page faults. As the program processes the data records, the program pages tend to replace some of the real pages assigned to the virtual buffer. In the non-partitioned experiments the double paging rate increases with increasing buffer size because for large virtual buffers the complete virtual buffer is rarely in real memory. The complete virtual buffer is rarely in memory due to competition for real memory by the program and the buffer and also due to the decrease in length of the string of requests relative to the buffer size.

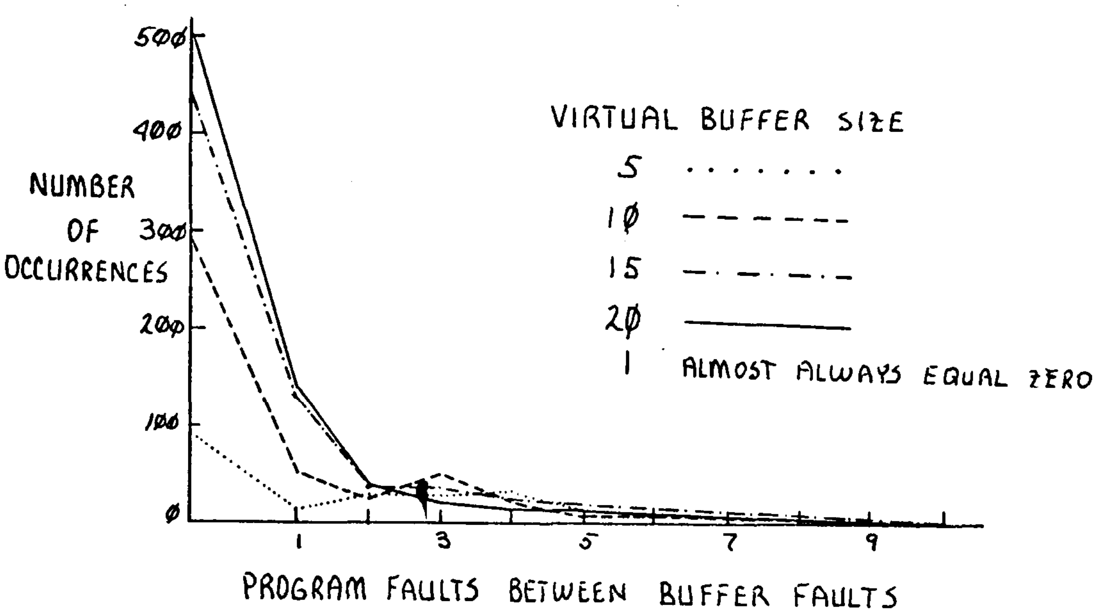

Fig. XII. Density functions of program faults between successive buffer faults for the SCH paging algorithm and SCH buffer manager using 36K of memory

The last component of buffer I/O is reference paging. The reference paging rates are very similar in the partitioned and non-partitioned experiments. The reference paging rates are an increasing function of virtual buffer size and a decreasing function of real memory size.

As shown in Figure I, the other component of system I/O is program paging. In the non-partitioned and partitioned experiments, program paging caused a large percentage of the paging in the system. In both sets of experiments the SCH paging algorithm clearly produced fewer page faults than the RAND paging algorithm. In the partitioned experiments there is no interaction between paging in the buffer and paging in the program. In the non-partitioned experiments, real pages were free to migrate between buffer space and program space and the migration is characterized in [2]. We define program paging in the non-partitioned experiments to consist of all page replacements caused by a fault in the program space. Figure XIII is a comparison of the program paging in the partitioned experiments and the non-partitioned experiments. The program partition size in the non-partitioned experiments is calculated by subtracting the average number of real pages allocated to the virtual buffer from the real memory size. As the real memory available to the program decreases, the program paging in the non-partitioned experiments has significantly fewer page faults than the program paging in the partitioned experiments. The lower program paging in the non-partitioned experiments reflects the alternating need for memory between the program and the buffer and the freedom of the pages in real memory to migrate between the program space and the buffer space.

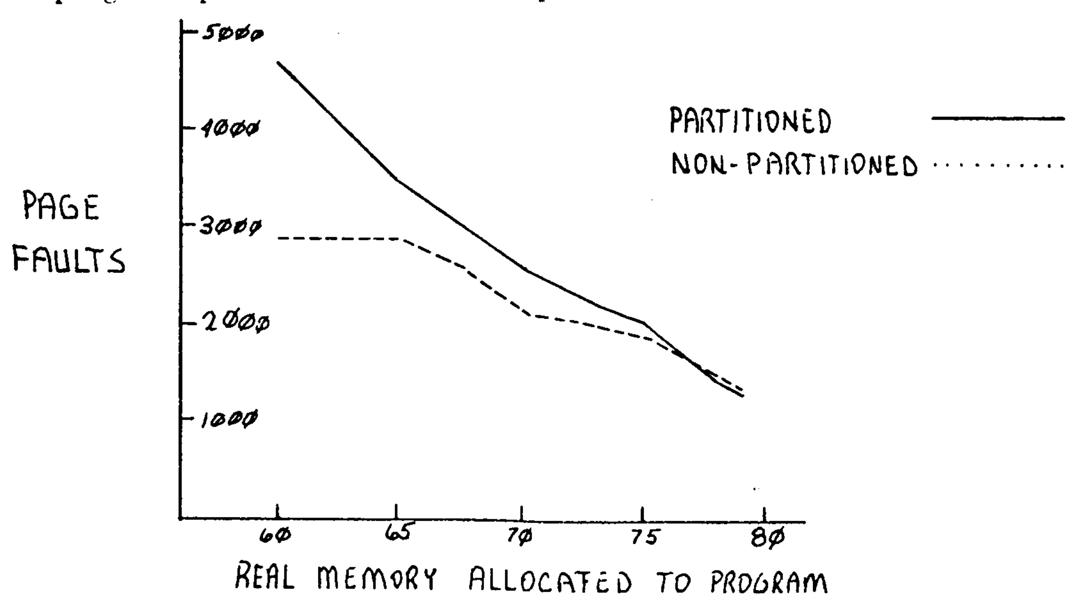

Fig. XIII. Program paging for the partitioned and non-partitioned experiments.

6. CONCLUSIONS

The performance of system I/O when the program space is not partitioned from the buffer space is usually better than the performance of system I/O in a partitioned system. We have shown that the system I/O in a partitioned system can be slightly lower than a non-partitioned system with a judicious choice of partition size although a poor choice of partition size will cause the system I/O in a partitioned system to be significantly higher. For a given virtual buffer size, the average number of real pages allocated to the virtual buffer in the non-partitioned experiments is a poor prediction of a buffer partition size to minimize system I/O.

The superior performance of a non-partitioned system was due to its ability to allocate real memory to the program space and buffer space as they alternated their requirements for memory. The allocation of real memory as required produces less program paging and a lower double paging rate in the buffer. When the virtual buffer size is large, the program paging in the non-partitioned system is much lower than the program paging in the partitioned system. As the virtual buffer size decreases, the difference in program paging performance decreases. When the virtual buffer size is small, the double paging rate in the non-partitioned system is much lower than the double paging rate in the partitioned system. As the virtual buffer size increases, the difference in double paging rate decreases but the corresponding decrease in I/O access for large buffers reduces the significance of the double paging rate.

Even though the performance of a partitioned system can be improved by using a different paging algorithm for the program partition and buffer partition, the gain in performance is not as significant as the performance improvement that was gained by not having a partition.

7. REFERENCES

1. Goldberg, R., and Hassinger, R., 'The Double Paging Anomaly', Proc. 1974 National Computer Conference, Chicago, May 6-8, 1974.

2. Sherman, S. W., and Brice, R. S., 'Performance of a Data Base Manager in a Virtual Memory System', to be published in ACM Transactions on Data Base Systems.

3. Brice, R. S., and Sherman, S. W., 'Performance of a Data Base Management System with Partially Locked Virtual Buffers', ICASE Report 76-6, Langley Research Center, Hampton, VA, March 1976.

4. Schwetman, H. D. and Browne, J. C., 'An Experimental Study of Computer System Performance', Proc. National ACM Conference, Boston, Mass., August 1972, pp. 693-703.

5. Tuel, W. G., 'An Analysis of Buffer Paging in Virtual Storage Systems', IBM Report RJ 1421, July 1974.

6. Tuel, W. G., 'An Analysis of Buffer Paging in Virtual Storage Systems', Proc. Third Texas Conference on Computing Systems, Austin, Texas, November 1974.

7. Hoare, C. A. R., and McKeag, R. M., 'A Survey of Store Management Techniques', A. P. I. C. Studies in Data Processing, No. 9, Academic Press, 1972.

8. Corbato, F. J., 'A Paging Experiment with the Multics System', In Honor of P. M. Morse, M.I.T. Press, Cambridge, Mass., 1969, pp. 217-228.

9. Grit, D. H., and Kain, R. Y., 'An Analysis of a Use Bit Page Replacement Algorithm', Proceedings ACM Annual Conference, 1975.

10. Spirn, J. R., and Denning, P. J. 'Experiments with Program Locality', Proc. FJCC, 1972, pp. 611-621.

11. Sherman, S. W., and Brice, R. S., 'I/O Buffer Performance in a Virtual Memory System', Proceedings Symposium on the Simulation of Computer Systems, Boulder Colo., August 1976.

Acknowledgement

We wish to thank Professor Jim Browne for suggesting this area of research and Professors Browne and Arden for their comments on our initial research. We also wish to acknowledge the technical support of Mary O. Smith